

Lecture 2

VHDL Refresher

Reading

Required

- P. Chu, *FPGA Prototyping by VHDL Examples*
Chapter 1, Gate-level combinational circuit

Recommended

- S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*
Chapter 2.10, Introduction to VHDL

Recommended reading

- Wikipedia – The Free On-line Encyclopedia

VHDL - <http://en.wikipedia.org/wiki/VHDL>

Verilog - <http://en.wikipedia.org/wiki/Verilog>

Brief History of VHDL



High Performance

CoolClock

Low Power

DataCache

VHDL

- VHDL is a language for describing digital hardware used by industry worldwide
- **VHDL** is an acronym for **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

A Brief History of VHDL

- July 1983: a DoD contract for the development of VHDL awarded to
 - Intermetrics
 - IBM
 - Texas Instruments
- August 1985: VHDL Version 7.2 released
- December 1987:
VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI standard

Subsequent versions of VHDL

- IEEE-1076 1987
- IEEE-1076 1993 ← most commonly supported by CAD tools
- IEEE-1076 2000 (minor changes)
- IEEE-1076 2002 (minor changes)
- IEEE-1076 2008

Verilog



High Performance

CoolClock

Low Power

Out-of-the-Box

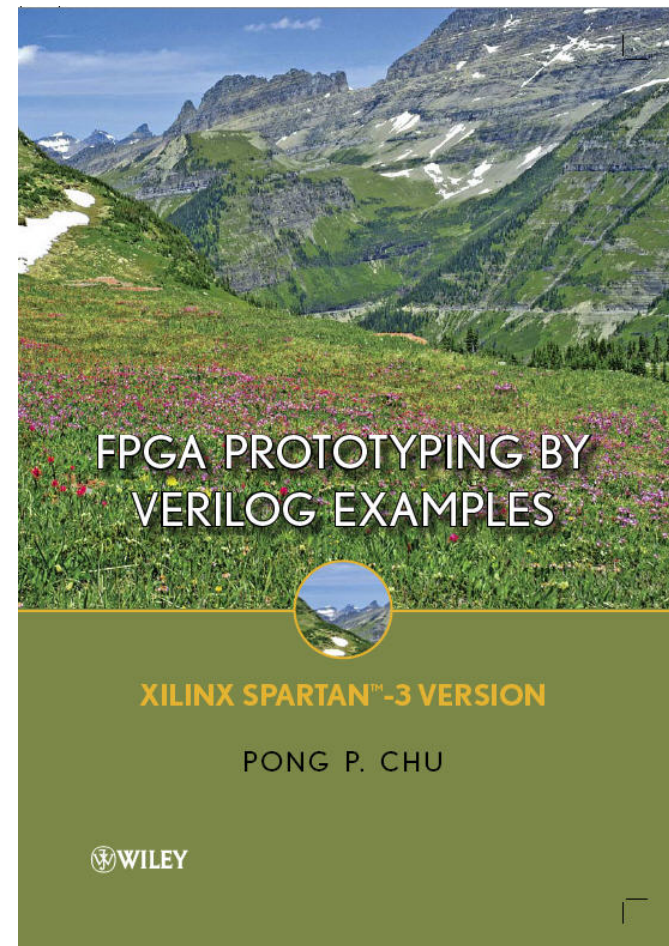
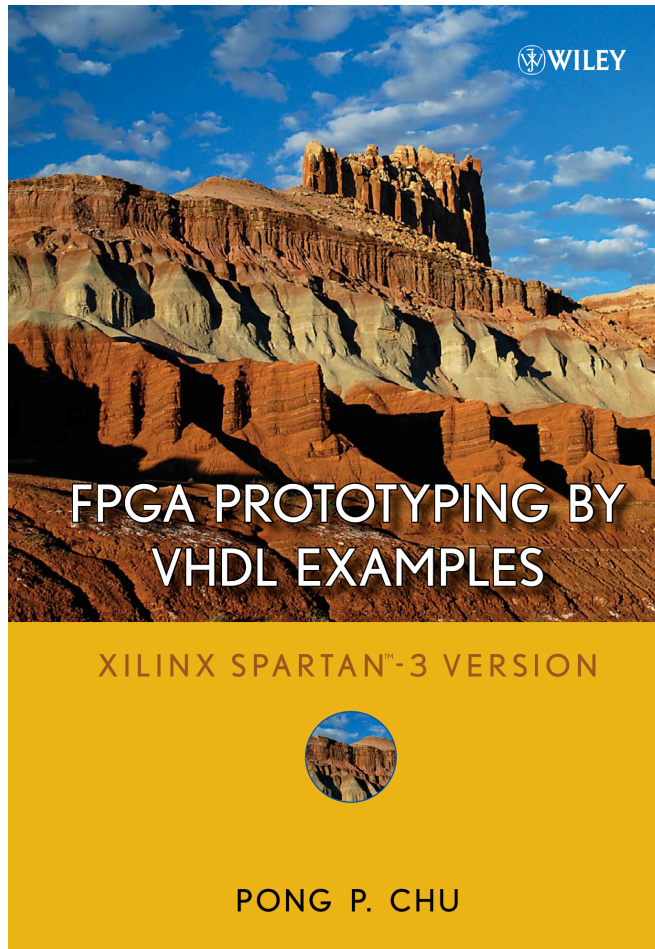
Verilog

- Essentially identical in function to VHDL
- Simpler and syntactically different
 - C-like
- Gateway Design Automation Co., 1985
- Gateway acquired by Cadence in 1990
- IEEE Standard 1364-1995 (Verilog-95)
- Early *de facto* standard for ASIC design
- Two subsequent versions
 - Verilog 2001 (major extensions) ← dominant version used in industry
 - Verilog 2005 (minor changes)
- Programming language interface to allow connection to non-Verilog code

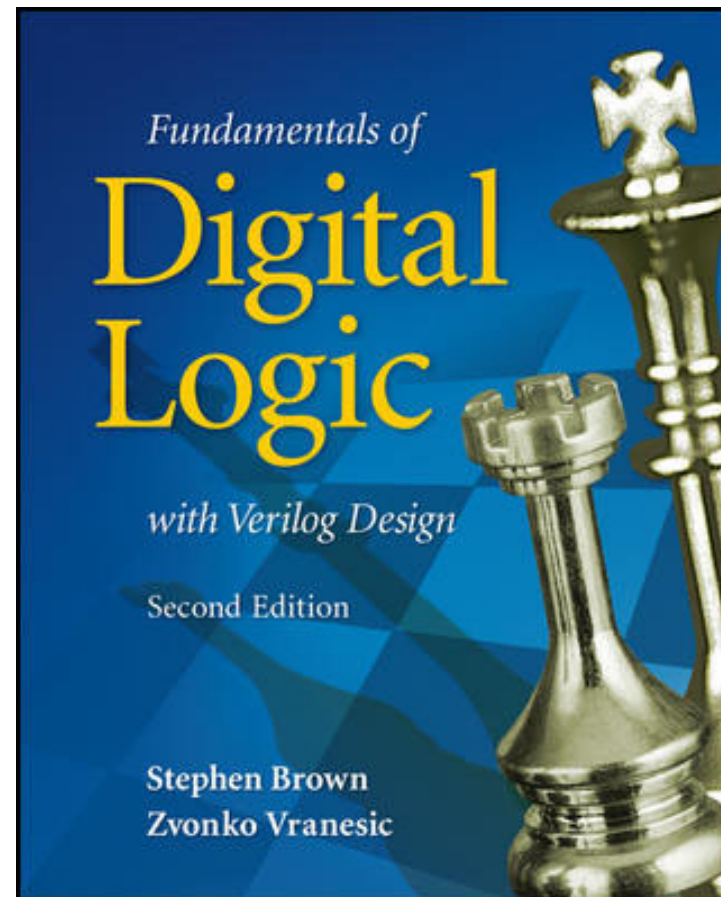
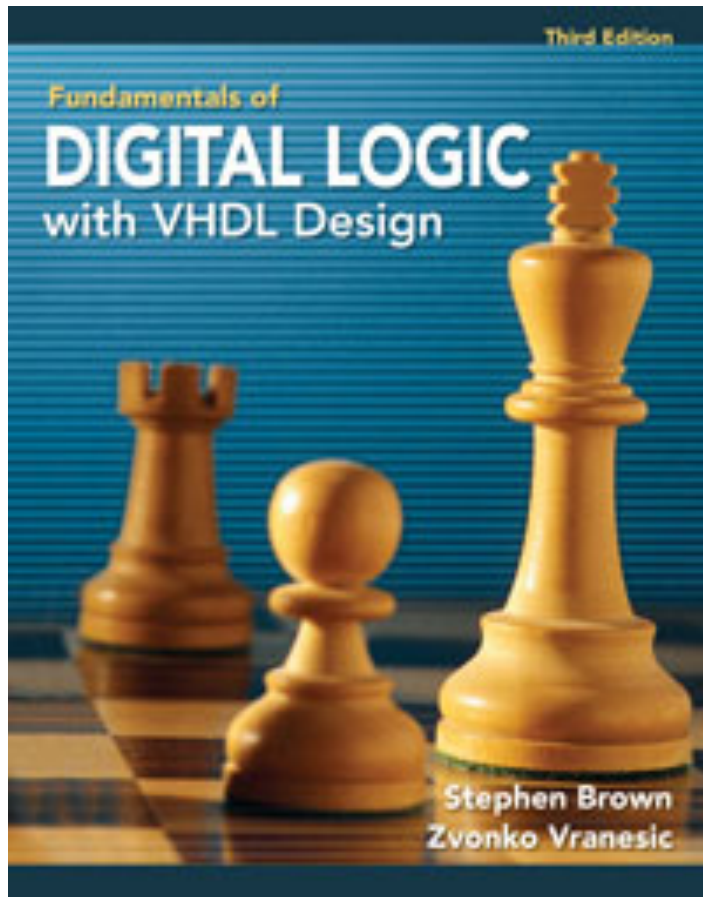
VHDL vs. Verilog

Government Developed	Commercially Developed
Ada based	C based
Strongly Type Cast	Mildly Type Cast
Case-insensitive	Case-sensitive
Difficult to learn	Easier to Learn
More Powerful	Less Powerful

How to learn Verilog by yourself ?



How to learn Verilog by yourself ?



Features of VHDL and Verilog

- Technology/vendor independent
- Portable
- Reusable

VHDL Fundamentals



Low Power

Low Power

Naming and Labeling (1)

- VHDL is case insensitive

Example:

Names or labels

databus

Databus

DataBus

DATABUS

are all equivalent

Naming and Labeling (2)

General rules of thumb (according to VHDL-87)

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (__) within a name (e.g., Sel__A is invalid)
5. All names and labels in a given entity and architecture must be unique

Free Format

- VHDL is a “free format” language

No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space and carriage return treated the same way.

Example:

```
if (a=b) then
```

or

```
if (a=b)           then
```

or

```
if (a =  
b) then
```

are all equivalent

Comments

- Comments in VHDL are indicated with a “double dash”, i.e., “--”
 - Comment indicator can be placed anywhere in the line
 - Any text that follows in the same line is treated as a comment
 - Carriage return terminates a comment
 - No method for commenting a block extending over a couple of lines

Examples:

-- main subcircuit

Data_in <= Data_bus; -- reading data from the input FIFO

Comments

- Explain Function of Module to Other Designers
- Explanatory, Not Just Restatement of Code
- Locate Close to Code Described
 - Put near executable code, not just in a header

Design Entity



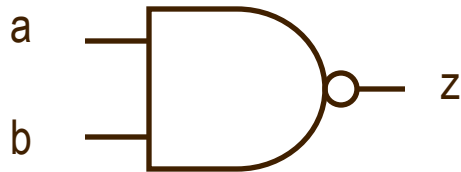
High Performance

CoolClock

Low Power

DataCache

Example: NAND Gate



a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

Example VHDL Code

- 3 sections to a piece of VHDL code
- File extension for a VHDL file is .vhd
- Name of the file **should be** the same as the entity name (nand_gate.vhd) [**OpenCores Coding Guidelines**]

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
ENTITY nand_gate IS
```

```
    PORT (
```

```
        a      : IN STD_LOGIC;
```

```
        b      : IN STD_LOGIC;
```

```
        z      : OUT STD_LOGIC);
```

```
END nand_gate;
```

```
ARCHITECTURE model OF nand_gate IS
```

```
BEGIN
```

```
    z <= a NAND b;
```

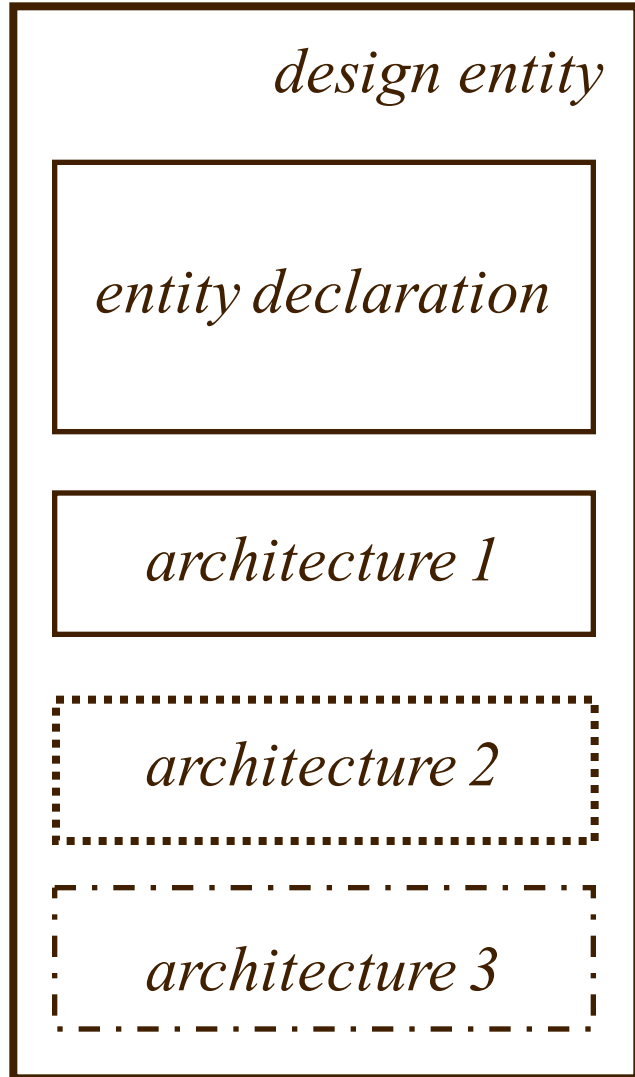
```
END model;
```

} LIBRARY DECLARATION

} ENTITY DECLARATION

} ARCHITECTURE BODY

Design Entity

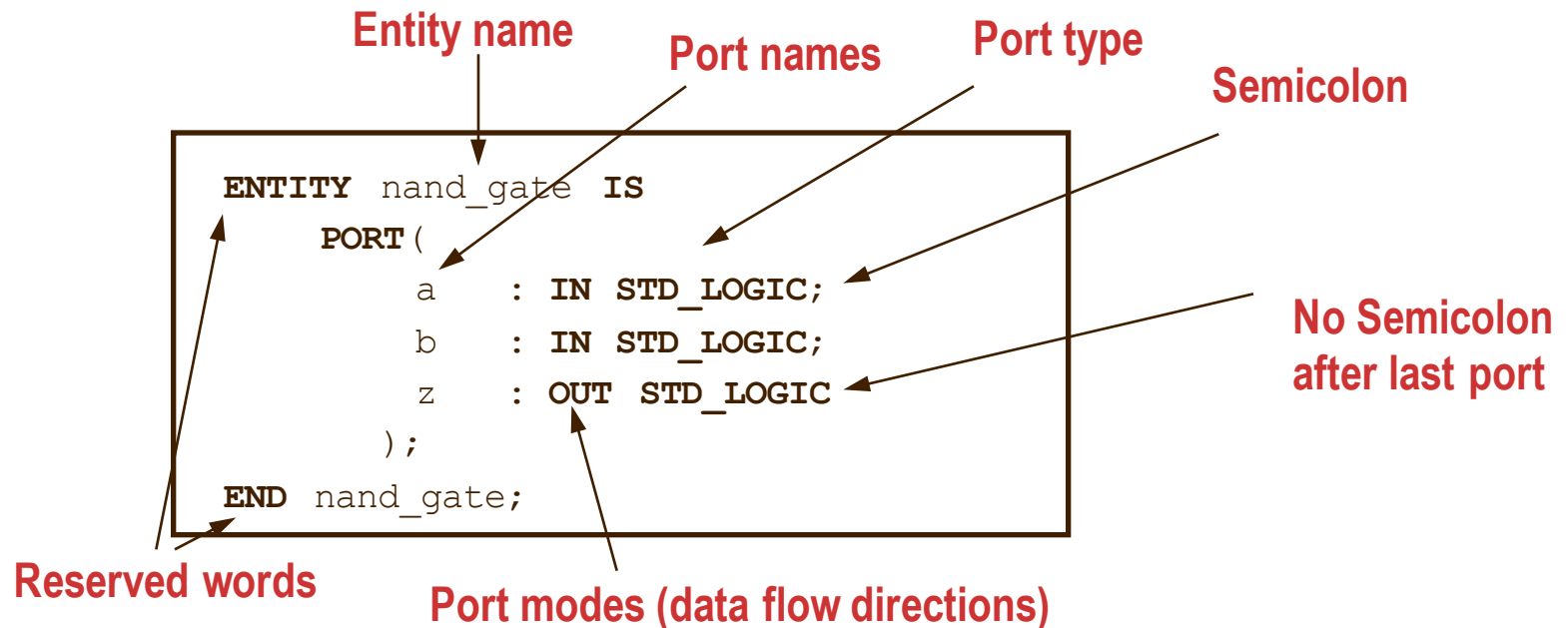


Design Entity - most basic building block of a design.

One *entity* can have many different *architectures*.

Entity Declaration

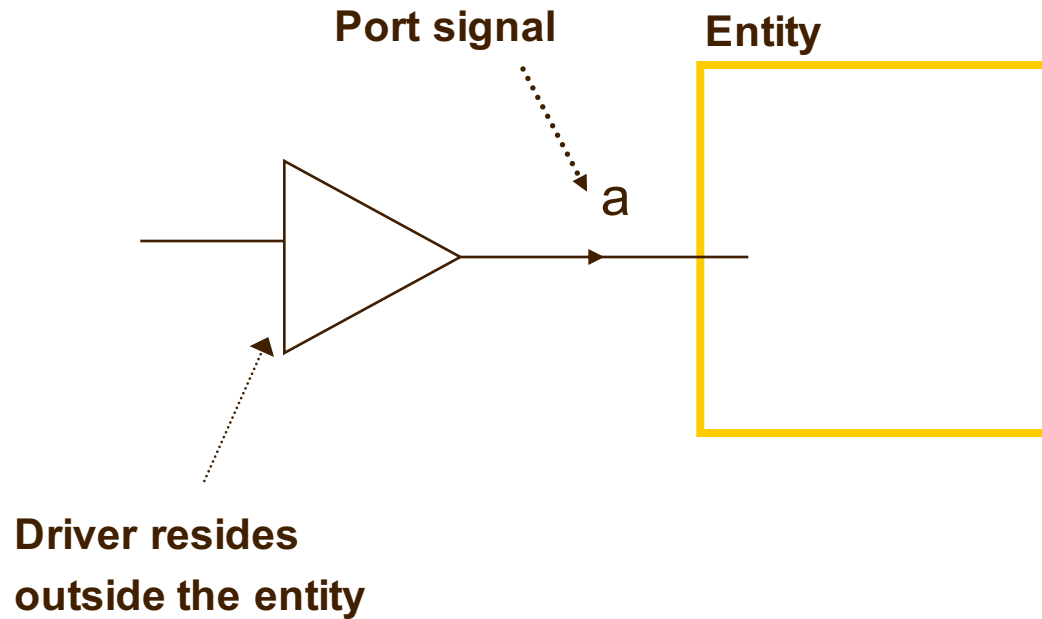
- Entity Declaration describes an interface of the component, i.e. input and output ports.



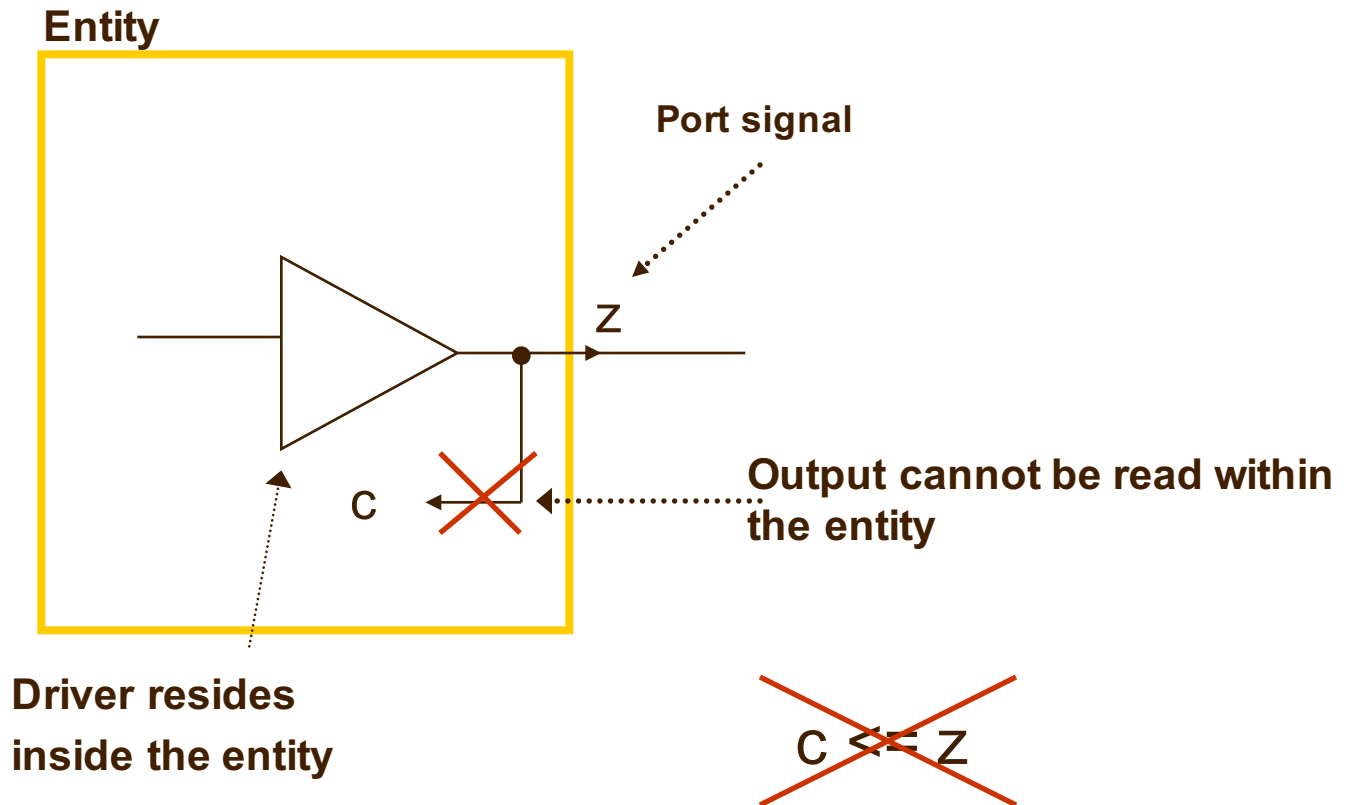
Entity declaration – simplified syntax

```
ENTITY entity_name IS  
  PORT (  
    port_name : port_mode signal_type;  
    port_name : port_mode signal_type;  
    .....  
    port_name : port_mode signal_type);  
END entity_name;
```

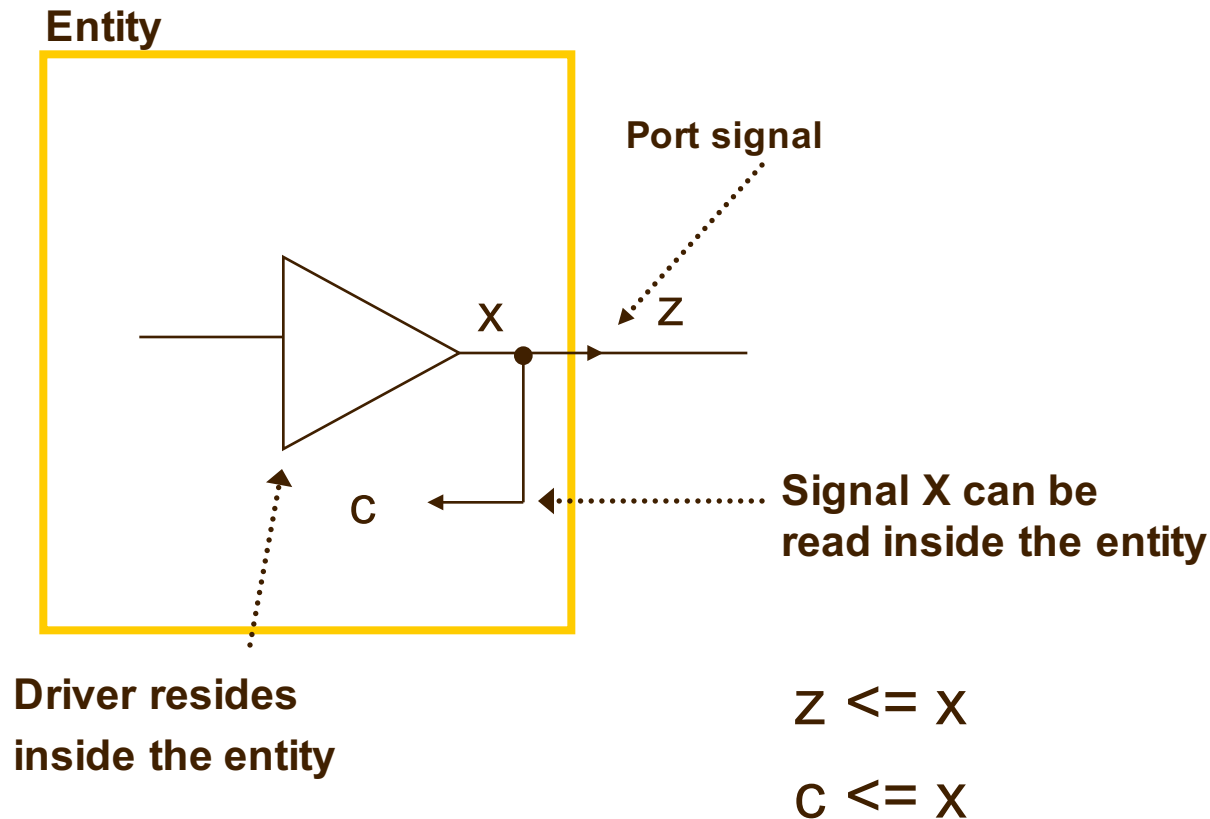
Port Mode IN



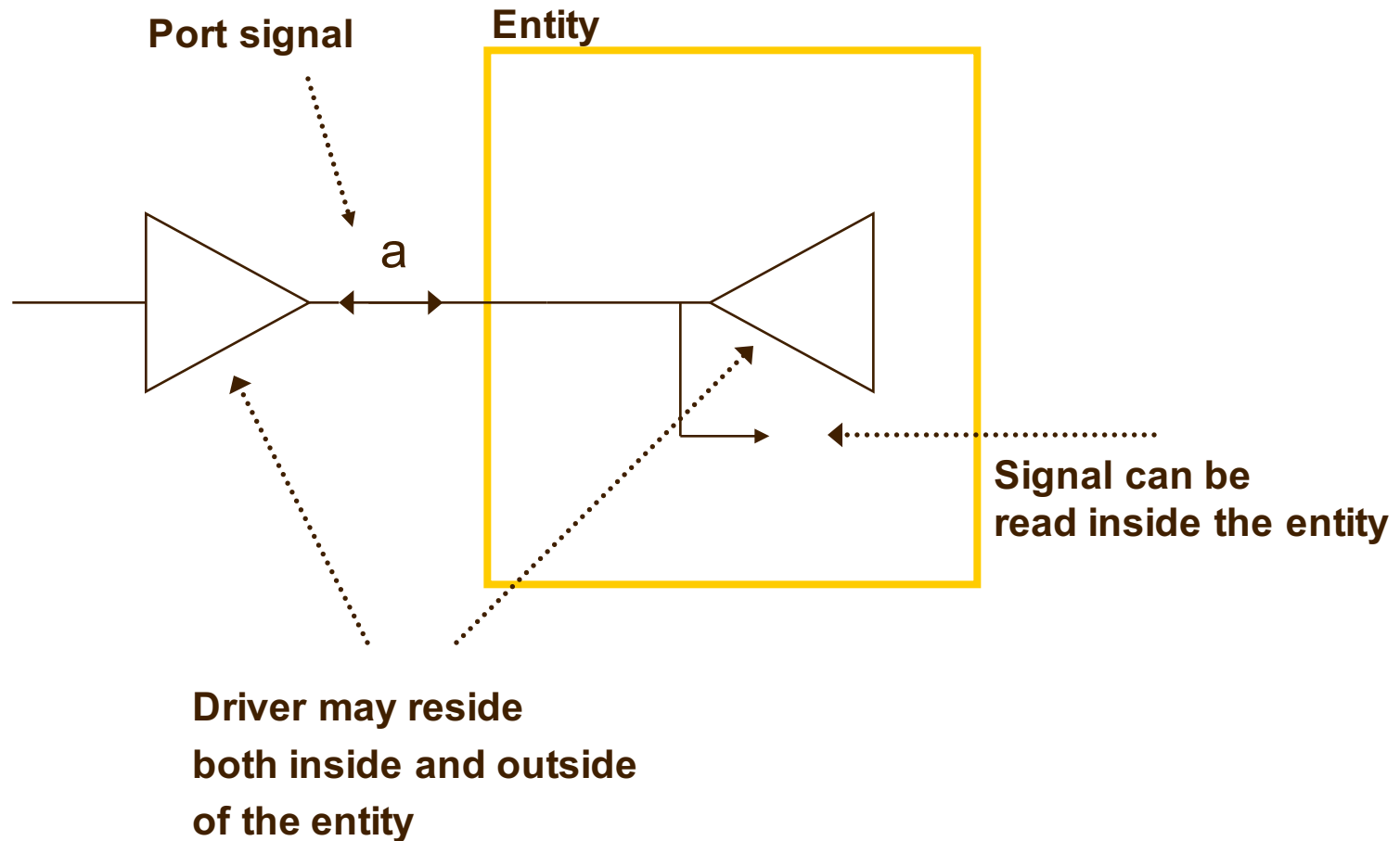
Port Mode OUT



Port Mode OUT (with extra signal)



Port Mode INOUT



Port Modes - Summary

The *Port Mode* of the interface describes the direction in which data travels with respect to the *component*

- **In:** Data comes into this port and can only be read within the entity. It can appear **only on the right side** of a signal or variable assignment.
- **Out:** The value of an output port can only be updated within the entity. **It cannot be read.** It can only appear **on the left side** of a signal assignment.
- **Inout:** The value of a bi-directional port can be read and updated within the entity model. It can appear on **both sides** of a signal assignment.

Architecture (Architecture body)

- Describes an implementation of a design entity
- Architecture example:

```
ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

Architecture – simplified syntax

```
ARCHITECTURE architecture_name OF entity_name IS  
    [ declarations ]  
BEGIN  
    code  
END architecture_name;
```

Entity Declaration & Architecture

nand_gate.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT (
        a    : IN STD_LOGIC;
        b    : IN STD_LOGIC;
        z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

Tips & Hints

Place each entity in a different file.

The name of each file should be exactly the same as the name of an entity it contains.

These rules are not enforced by all tools but are worth following in order to increase readability and portability of your designs

Tips & Hints

Place the declaration of each port,
signal, constant, and variable
in a separate line

These rules are not enforced by all tools
but are worth following in order to increase
readability and portability of your designs



Libraries

High Performance

CoolClock

Low Power

DataCache

Library Declarations

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT (
        a    : IN STD_LOGIC;
        b    : IN STD_LOGIC;
        z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

Library declaration

Use all definitions from the package
std_logic_1164

Library declarations - syntax

```
LIBRARY library_name;  
USE library_name.package_name.package_parts;
```

Fundamental parts of a library

LIBRARY

PACKAGE 1

TYPES
CONSTANTS
FUNCTIONS
PROCEDURES
COMPONENTS

PACKAGE 2

TYPES
CONSTANTS
FUNCTIONS
PROCEDURES
COMPONENTS

Libraries

- **ieee**

Specifies multi-level logic system,
including STD_LOGIC, and
STD_LOGIC_VECTOR data types

} Need to be explicitly
declared

- **std**

Specifies pre-defined data types
(BIT, BOOLEAN, INTEGER, REAL,
SIGNED, UNSIGNED, etc.), arithmetic
operations, basic type conversion
functions, basic text i/o functions, etc.

} Visible by default

- **work**

Holds current designs after compilation

STD_LOGIC Demystified



High Performance

CoolClock

Low Power

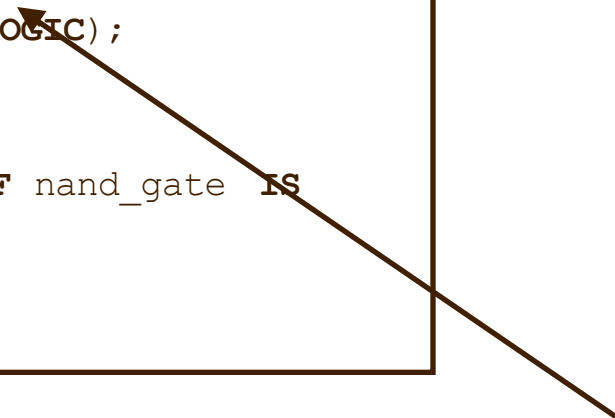
DataCache

STD_LOGIC

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT (
        a    : IN STD_LOGIC;
        b    : IN STD_LOGIC;
        z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```



What is **STD_LOGIC** you ask?

BIT versus STD_LOGIC

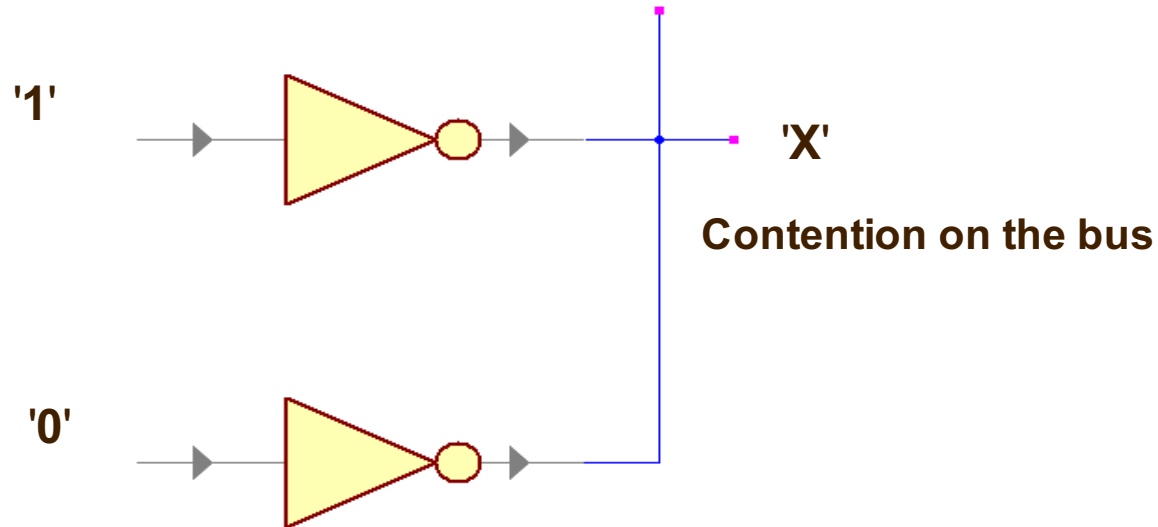
- BIT type can only have a value of '0' or '1'
- STD_LOGIC can have nine values
 - '0', '1', 'Z', 'U', 'X', 'L', 'H', 'W', '-'
Useful mainly for simulation
 - '0', '1', and 'Z' are synthesizable
(**your codes should contain only these three values**)

STD_LOGIC *type* demystified

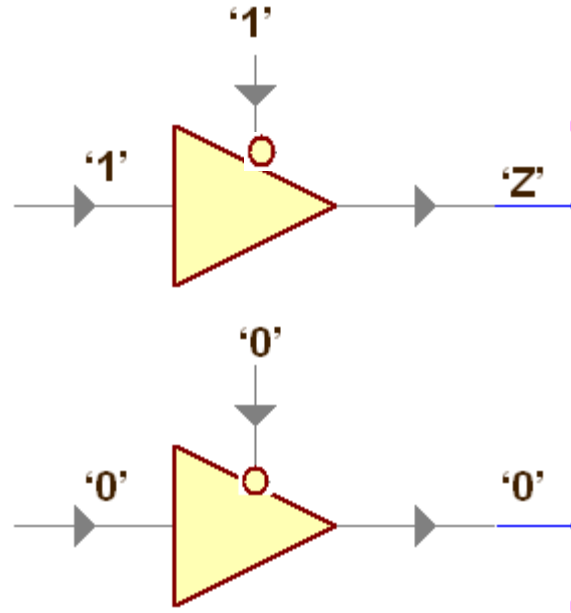
Value	Meaning
'U'	Uninitialized
'X'	Forcing (Strong driven) Unknown
'0'	Forcing (Strong driven) 0
'1'	Forcing (Strong driven) 1
'Z'	High Impedance
'W'	Weak (Weakly driven) Unknown
'L'	Weak (Weakly driven) 0. Models a pull down.
'H'	Weak (Weakly driven) 1. Models a pull up.
'-'	Don't Care

More on STD_LOGIC Meanings (1)

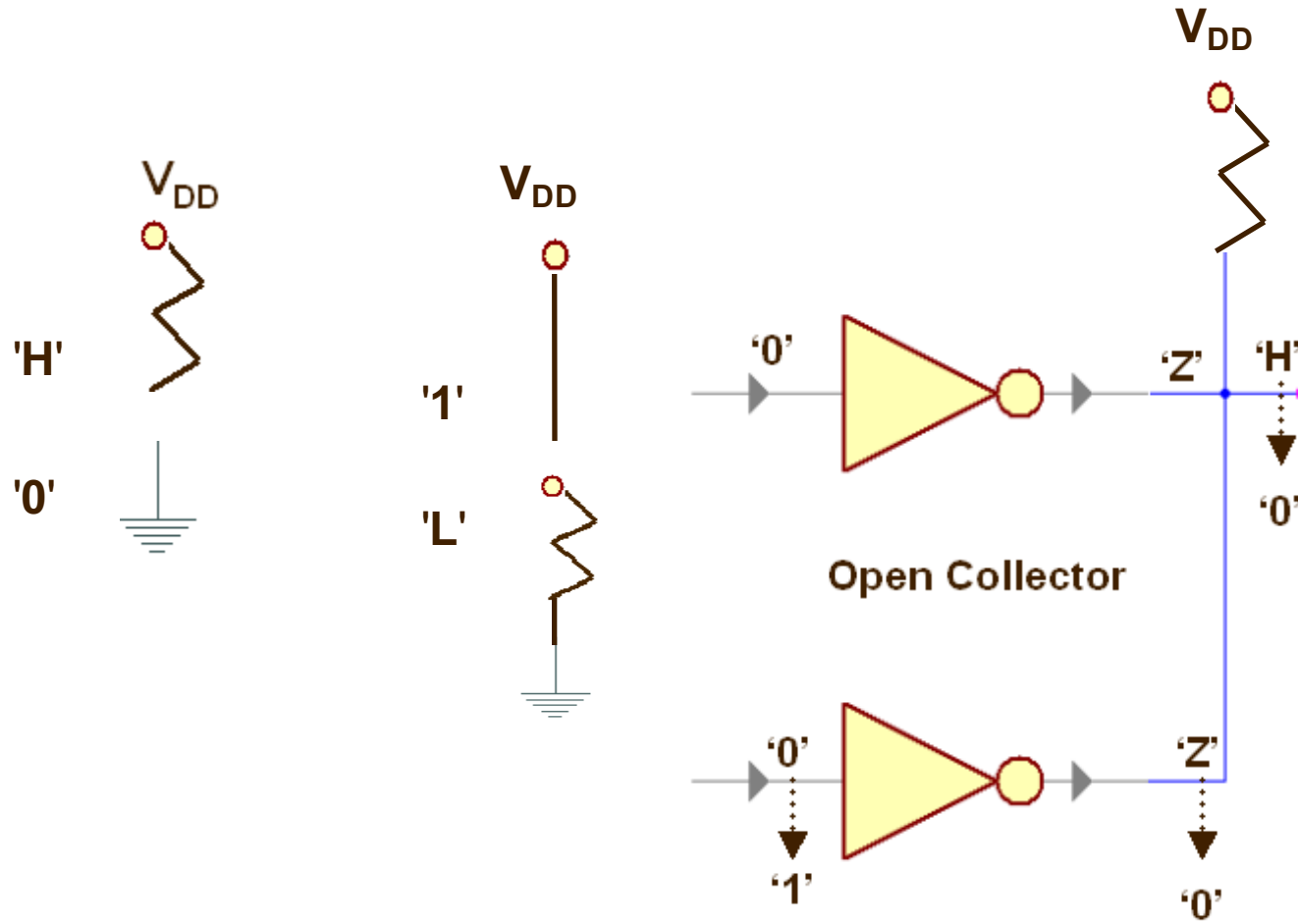
X



More on STD_LOGIC Meanings (2)



More on STD_LOGIC Meanings (3)

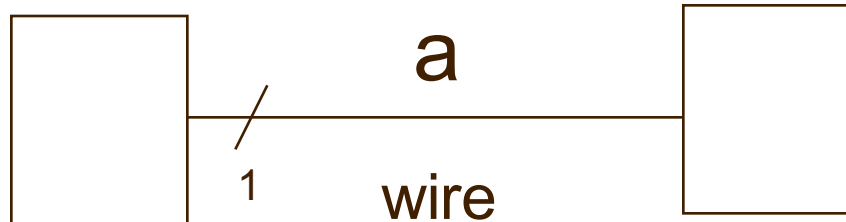


Modeling Wires and Buses

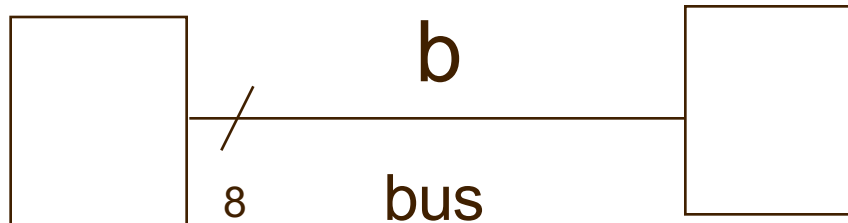


Signals

SIGNAL a : STD_LOGIC;



SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



Standard Logic Vectors

```
SIGNAL a: STD_LOGIC;  
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNT0 0);  
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNT0 0);  
SIGNAL d: STD_LOGIC_VECTOR(15 DOWNT0 0);  
SIGNAL e: STD_LOGIC_VECTOR(8 DOWNT0 0);  
  
.....  
  
a <= ' 1' ;  
b <= "0000";           -- Binary base assumed by default  
c <= B"0000";          -- Binary base explicitly specified  
d <= X"AF67";          -- Hexadecimal base  
e <= O"723";           -- Octal base
```

Vectors and Concatenation

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNT0 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNT0 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNT0 0);

a <= "0000";
b <= "1111";
c <= a & b;           -- c = "00001111"

d <= '0' & "0001111";  -- d <= "00001111"

e <= '0' & '0' & '0' & '0' & '1' & '1' &
    '1' & '1';
                        -- e <= "00001111"
```

Types of VHDL Description (Modeling Styles)



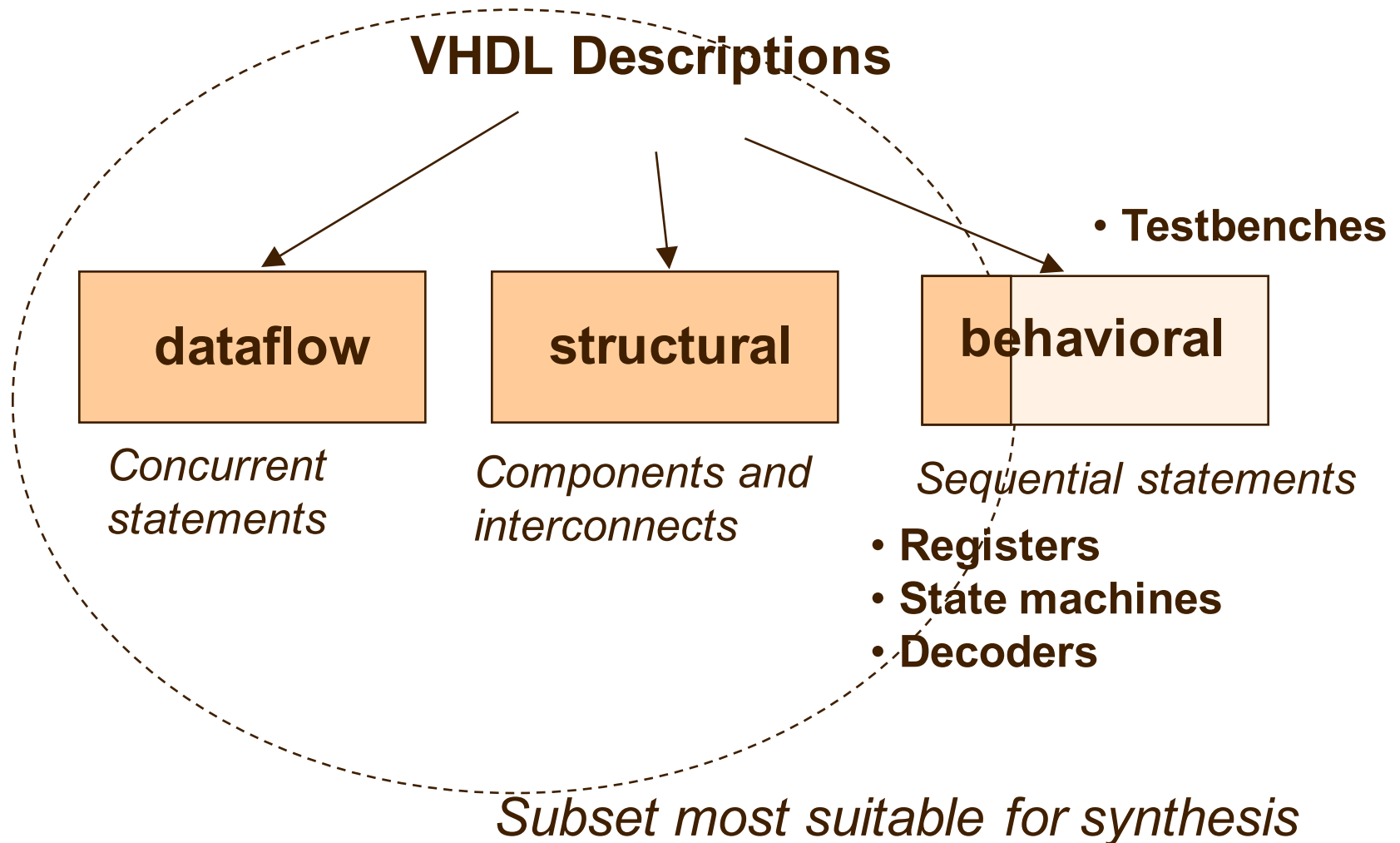
Low Power

Portability

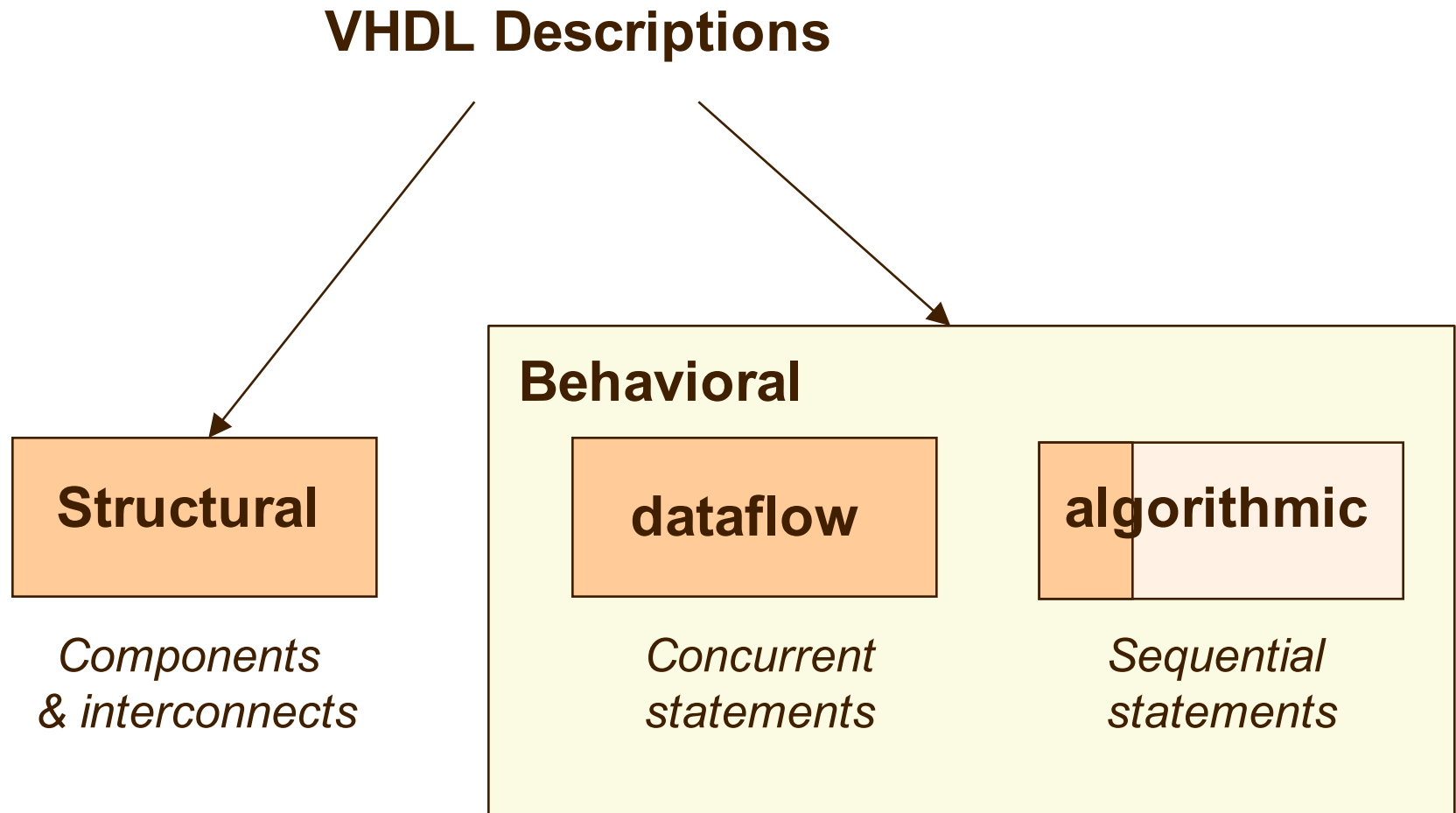
CoolClock

High Performance

Types of VHDL Description: Convention used in this class



Types of VHDL Description: Alternative convention



xor3 Example



Entity xor3_gate

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY xor3_gate **IS**

PORT(

A : **IN** STD_LOGIC;

B : **IN** STD_LOGIC;

C : **IN** STD_LOGIC;

Result : **OUT** STD_LOGIC

);

end xor3_gate;

Dataflow Architecture (xor3_gate)

ARCHITECTURE dataflow **OF** xor3_gate **IS**
SIGNAL U1_OUT: STD_LOGIC;
BEGIN
 U1_OUT <= A **XOR** B;
 Result <= U1_OUT **XOR** C;
END dataflow;



Dataflow Description

- Describes how data moves through the system and the various processing steps.
 - Dataflow uses series of concurrent statements to realize logic.
 - Dataflow is most useful style when series of Boolean equations can represent a logic → used to implement simple combinational logic
 - Dataflow code also called “concurrent” code
- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn't matter
 - This is not true for sequential/behavioral statements

This order...

```
U1_out <= A XOR B;  
Result <= U1_out XOR C;
```

Is the same as this order...

```
Result <= U1_out XOR C;  
U1_out <= A XOR B;
```

Structural Architecture in VHDL 93

```
ARCHITECTURE structural OF xor3_gate IS  
SIGNAL U1_OUT: STD_LOGIC;
```

```
BEGIN
```

```
U1: entity work.xor2(dataflow)
```

```
  PORT MAP (I1 => A,
```

```
            I2 => B,
```

```
            Y => U1_OUT);
```

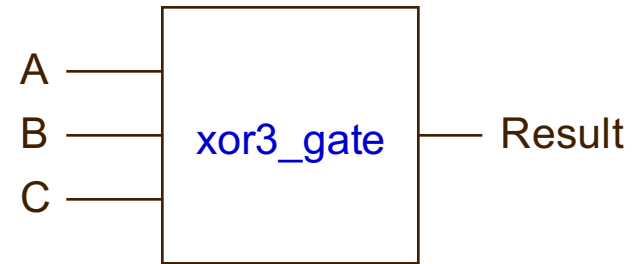
```
U2: entity work.xor2(dataflow)
```

```
  PORT MAP (I1 => U1_OUT,
```

```
            I2 => C,
```

```
            Y => Result);
```

```
END structural;
```



PORT NAME

LOCAL WIRE NAME

xor2

xor2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor2 IS
    PORT (
        I1    : IN STD_LOGIC;
        I2    : IN STD_LOGIC;
        Y      : OUT STD_LOGIC);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
    Y <= I1 xor I2;
END dataflow;
```

Structural Architecture in VHDL 87

```
ARCHITECTURE structural OF xor3_gate IS  
SIGNAL U1_OUT: STD_LOGIC;
```

```
COMPONENT xor2
```

```
  PORT(  
    I1 : IN STD_LOGIC;  
    I2 : IN STD_LOGIC;  
    Y : OUT STD_LOGIC
```

```
  );
```

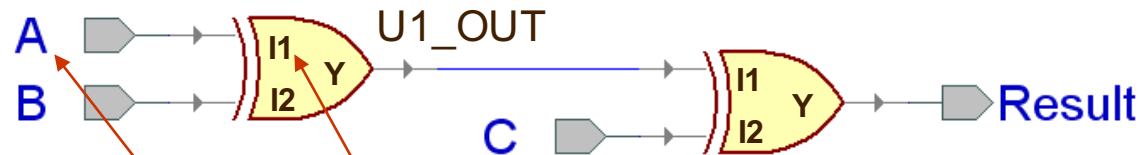
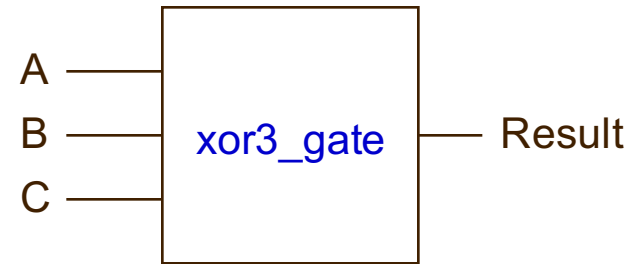
```
END COMPONENT;
```

```
BEGIN
```

```
  U1: xor2 PORT MAP (I1 => A,  
                     I2 => B,  
                     Y => U1_OUT);
```

```
  U2: xor2 PORT MAP (I1 => U1_OUT,  
                     I2 => C,  
                     Y => Result);
```

```
END structural;
```



Structural Description

- Structural design is the simplest to understand. This style is the closest to schematic capture and utilizes simple building blocks to compose logic functions.
- Components are interconnected in a hierarchical manner.
- Structural descriptions may connect simple gates or complex, abstract components.
- Structural style is useful when expressing a design that is naturally composed of sub-blocks.

Behavioral Architecture (xor3 gate)

ARCHITECTURE behavioral **OF** xor3 **IS**
BEGIN

xor3_behave: **PROCESS** (A, B, C)

BEGIN

IF ((A **XOR** B **XOR** C) = '1') **THEN**

 Result <= '1';

ELSE

 Result <= '0';

END IF;

END PROCESS xor3_behave;

END behavioral;

Behavioral Description

- It accurately models what happens on the inputs and outputs of the black box (no matter what is inside and how it works).
- This style uses PROCESS statements in *VHDL*.

